Перегрузка операторов
«Дружественная» операторная функция

Существует два способа перегрузки любого оператора:

- с помощью операторной функции, которая реализована внутри класса;
- с помощью операторной функции, которая реализована как «дружественная» (friend) к классу.

Между способами реализации операторных функций существует отличие в количестве параметров, которые получает операторная функция:

- для унарных операторов операторная функция внутри класса не получает параметров. А «дружественная» к классу операторная функция получает один параметр;
- для бинарных операторов операторная функция внутри класса получает один параметр. А «дружественная» функция получает два параметра. В этом случае первым параметром «дружественной» функции есть левый операнд, а вторым параметром правый операнд.

Эти отличия возникают из-за того, что «дружественная» операторная функция не получает неявного указателя this. Поэтому, в ней нужно явным образом задавать параметры.

Форма операторной функции, реализованной за пределами класса («дружественная» функция к классу)

Операторная функция может быть реализована за пределами класса. Если операторная функция перегружает унарный оператор, то она содержит один параметр. Если операторная функция перегружает бинарный оператор, то она содержит два параметра.

Общая форма операторной функции, которая есть дружественной к классу имеет вид

```
return_type operator#(arguments_list)
{
   // некоторые операции
   // ...
}
```

где

- return_type – тип значения, которое возвращает операторная функция;
- operator# – ключевое слово, которое определяет операторную функцию в классе. Символ # определяет оператор языка С++, который перегружается. Например, если перегружается оператор +, то нужно указать operator+;
- argument_list – список параметров, которые получает операторная функция. Если в «дружественной» функции перегружается бинарный оператор, то argument_list содержит два аргумента. Если перегружается унарный оператор, то argument_list содержит один аргумент.

В классе эта функция должна быть объявлена как «дружественная» с ключевым словом friend.

Перегрузка бинарного оператора ‘–’ как «дружественная» операторная функция классу

По данному примеру можно разрабатывать собственные операторные функции, которые являются «дружественными» к заданному классу.

Задан класс Complex, реализующий комплексное число. В классе объявляются внутренние переменные, конструкторы, методы доступа и «дружественная» функция operator-(). «Дружественная» функция operator-(), реализованная за пределами класса, осуществляет вычитание комплексных чисел.

```
// класс Complex
class Complex
{
```

```
private:
   float real; // вещественная часть
   float imag; // мнимая часть

public:
   // конструкторы
   Complex(void)
   {
      real = imag = 0;
   }

   Complex(float _real, float _imag)
   {
      real = _real;
      imag = _imag;
   }

   // методы доступа
   float GetR(void) { return real; }
   float GetI(void) { return imag; }

   void SetRI(float _real, float _imag)
   {
      real = _real;
      imag = _imag;
   }

   // объявление "дружественной" к классу Complex операторной функции
   friend Complex operator-(Complex c1, Complex c2);
};

// "дружественная" к классу Complex операторная функция,
// реализована за пределами класса,
// осуществляет вычитание комплексных чисел
Complex operator-(Complex c1, Complex c2)
{
   Complex c; // создать объект класса Complex

   // вычитание комплексных чисел
   c.real = c1.real - c2.real;
   c.imag = c1.imag - c2.imag;

   return c;
}
Использование класса Complex в другом методе
// использование "дружественной" операторной функции
Complex c1(5,6);
Complex c2(3,-2);
Complex c3; // результат
float a, b;

// проверка
```

```
a = c1.GetR(); // a = 5
b = c1.GetI(); // b = 6

// вызов "дружественной" к классу Complex операторной функции
c3 = c1 - c2;

// результат
a = c3.GetR(); // a = 5-3 = 2
b = c3.GetI(); // b = 6-(-2) = 8
```

Как видно из вышеприведенного примера, «дружественная» функция operator-() получает два параметра. Первый параметр соответствует левому операнду в бинарном операторе вычитания '–'. Второй параметр соответствует правому операнду.

Перегрузка оператора '/'. Реализация класса, содержащего динамический массив вещественных чисел. Операторная функция реализована как «дружественная» функция

В примере демонстрируется, как правильно реализовывать перегрузку «дружественных» операторных функций для классов, в которых память выделяется динамически. Класс содержит динамический массив элементов типа float.

Объявляется класс ArrayFloat, который реализует динамический массив вещественных чисел. В классе реализованы следующие элементы:

- внутренняя переменная size – размер массива;
- указатель A на тип float. Это есть массив, память для которого выделяется динамически;
- конструктор класса ArrayFloat() без параметров;
- конструктор с двумя параметрами ArrayFloat(int, float*);
- конструктор копирования ArrayFloat(const ArrayFloat&). Этот конструктор необходим для избежания недостатков побитового копирования. Без конструктора копирования работа объектов класса будет иметь ошибки;
- деструктор ~ArrayFloat(). В деструкторе освобождается память, выделенная для динамического массива A;
- методы GetSize(), SetSize() для доступа к переменной size;
- методы GetAi(), SetAi() для доступа к конкретному элементу массива с заданным индексом;
- метод FormArray(), формирующий элементы массива по формуле;
- метод Print(), который выводит массив;
- операторная функция operator=(const ArrayFloat&). Эта функция перегружает оператор присваиания =, который реализует копирование объектов. Функция необходима для избежания недостатков побитового копирования в случае присваивания объектов obj1=obj2;
- «дружественная» к классу ArrayFloat операторная функция operator/(const ArrayFloat&, const ArrayFloat&), которая перегружает оператор деления / для данного класса. Функция реализует поэлементное деление массивов. Если количество элементов массивов не совпадает, то результирующий массив устанавливается в размер наименьшего из двух массива.

Реализация класса для приложения типа Console Application следующая:

```
#include <iostream>
using namespace std;

// клас - массив типа float
class ArrayFloat
{
private:
```

```
  int size;
  float * A; // динамический размер массива

public:
  // конструкторы класса
  // конструктор без параметров
  ArrayFloat()
  {
    size = 0;
    A = nullptr;
  }

  // конструктор с двумя параметрами
  ArrayFloat(int nsize, float * nA)
  {
    size = nsize;
    A = new float[size];

    for (int i = 0; i < nsize; i++)
      A[i] = nA[i];
  }

  // конструктор копирования,
  // нужен для избежания недостатков побитового копирования
  ArrayFloat(const ArrayFloat& nA)
  {
    if (size > 0)
    {
      delete[] A; // освободить предварительно выделенную память
    }

    // выделить память для A
    A = new float[nA.size];

    // выполнить копирование *this <= nA
    size = nA.size;
    for (int i = 0; i < size; i++)
      A[i] = nA.A[i];
  }

  // деструктор
  ~ArrayFloat()
  {
    // освободить память, выделенную для массива A
    if (size > 0)
      delete[] A;
  }

  // методы доступа
  int GetSize(void) { return size; }

  void SetSize(int nsize)
```

```
{
  if (size > 0)
    delete[] A;

  size = nsize;
  A = new float[size]; // выделить новый фрагмент памяти

  // заполнить массив нулями
  for (int i = 0; i < size; i++)
    A[i] = 0.0f;
}

// считать значение по індексу index
float GetAi(int index)
{
  if ((index >= 0) && (index < size))
    return A[index];
  else
    return 0;
}

// установить новое значение
void SetAi(int index, float value)
{
  if ((index >= 0) && (index < size))
    A[index] = value;
}

// метод, формирующий массив
void FormArray()
{
  for (int i = 0; i < size; i++)
    A[i] = (float)i;
}

// метод, отображающий массив
void Print(const char* objName)
{
  cout << "Object: " << objName << endl;
  for (int i = 0; i < size; i++)
    cout << A[i] << " ";
  cout << endl << endl;
}

// Перегрузка операторов
// Перегрузка оператора копирования operator=(const ArrayFloat&)
ArrayFloat operator=(const ArrayFloat& nA)
{
  if (size > 0)
    delete[] A; // освободить ранее выделенную память

  // выделить память для A заново
```

```
    A = new float[nA.size];

    // выполнить копирование *this <= nA
    size = nA.size;
    for (int i = 0; i < size; i++)
      A[i] = nA.A[i];
    return *this;
  }

  // "дружественная" операторная функция '/',
  // только объявление, реализация за пределами класса
  friend ArrayFloat operator/(const ArrayFloat& A1, const ArrayFloat& A2);
};

// "дружественная" к классу ArrayFloat операторная функция operator/()
// функция получает два параметра A1, A2
ArrayFloat operator/(const ArrayFloat& A1, const ArrayFloat& A2)
{
  // создать объект класса ArrayFloat
  ArrayFloat A; // вызывается конструктор без параметров
  int n;

  // взять минимальное значение из размера двух массивов A1, A2
  n = A1.size;
  if (n > A2.size) n = A2.size;

  // установить новый размер массива A, перераспределение памяти
  A.SetSize(n);

  // поэлементное деление
  for (int i = 0; i < n; i++)
  {
    if (A2.A[i] != 0) // обойти деление на 0
      A.A[i] = A1.A[i] / A2.A[i];
    else
      A.A[i] = 0.0f;
  }

  return A; // вернуть новый объект
}

void main()
{
  // Демонстрация работы класса ArrayFloat
  ArrayFloat AF1; // конструктор без параметров
  AF1.FormArray();

  int size; // дополнительные переменные
  float x;

  size = AF1.GetSize(); // size = 0
  x = AF1.GetAi(3); // x = 0
```

```
 cout << "size = " << size << endl;
 cout << "x = " << x << endl;

 ArrayFloat AF2 = AF1; // вызов конструктора копирования
 cout << AF2.GetAi(3) << endl;

 ArrayFloat AF3;
 AF3.SetSize(10);
 AF3.FormArray();
 AF2 = AF3; // вызов оператора копирования operator=()
 cout << "AF2[3] = " << AF2.GetAi(3) << endl;

 ArrayFloat AF4 = AF2; // вызов конструктора копирования
 cout << "AF4[5] = " << AF2.GetAi(5) << endl;

 // оператор копирования в виде "цепочки"
 AF1 = AF2 = AF4 = AF3; // вызов оператора копирования operator=()
 cout << "AF1[8] = " << AF1.GetAi(8) << endl;

 // Демонстрация "дружественной" операторной функции operator/
 AF3 = AF2 / AF1; // вызов "дружественной" функции operator/()
 cout << endl << endl;
 AF3.Print("AF3");

 // "Дружественная" функция работает в виде "цепочки"
 ArrayFloat AF5;
 AF5 = AF1 / AF2 / AF3;
 cout << endl;
 AF5.Print("AF5");
}
```

Результат работы программы:

```
size = 0
x = 0
0
AF2[3] = 3
AF4[5] = 5
AF1[8] = 8


Object: AF3
0 1 1 1 1 1 1 1 1 1


Object: AF5
0 1 1 1 1 1 1 1 1 1

Object: AF6
1.1 2.3 4.5
```